SOFT8023 – Lab 1

# Overview

You will begin by creating an initial application architecture that will be improved as the weeks progress. The application will simulate a battle royale of ordinary people, superheroes and villains (think of something like the Royal Rumble in WWE wrestling). It will become part of your first assignment submission.

# Step 1

You are to make good use of the object-oriented programming principles you have learned so far, such as polymorphism, interfaces, enumerations and more. You have some flexibility in your approach, but there are some recommendations below.

I want to see well-structured projects, so please use packages and clean code. I like the reverse domain name convention for packages, e.g. your base package might be ie.cit.soft8023.myapp. You might put your main application classes there and domain classes in a sub-package, such as ie.cit.soft8023.myapp.domain.

Create the following domain classes with at least the listed properties (you can add more if they make sense):

* Being (Abstract class)
  + Name
  + Type
  + Core Attributes: health, stamina, power, speed
* HumanBeing inheriting from Being
  + There should be a link to a super being if the human transforms into a super being, e.g. Bruce Banner turning into the Hulk
* SuperBeing inheriting from Being
  + There should be a link to a human being if the super being transformed from a human
* BaseAttack (Abstract class) implements an Attack interface (with method, *attack*)
* A number of subclasses of BaseAttack, e.g.
  + Punch
  + Kick
  + FreezeRay
  + LaserEye, etc.

Attack implementations, which override the *attack* method from the Attack interface, should return an object containing adjustments to the core attributes of health, stamina, power and speed (e.g. a freeze ray might reduce speed to zero – this might factor into your fight algorithm later). Only non-super powers can be assigned to humans, such as punch and kick, but not laser eye or freeze ray, etc. Each instance of an attack can have a multiplier, e.g. A Mega Mutant might have a 3 x multiplier for punching and kicking (whereas it may be just 1 for a human).

To distinguish between human being, super beings or any other type of being, an enumeration is recommended, i.e. using an Enum type (e.g. called BeingType). A property of this type in the base Being class could be useful for later decisions.

I recommend using constructors and having additional overloaded constructors where it makes sense. E.g. you might have a standard constructor for a SuperBeing that accepts initial values, such as name and core attributes, but you might have another that also accepts a human being reference if the super being transforms from a human.

Beings should identify themselves in three ways:

* What is your name?
* What are you? (e.g. “I am a human being!” or “I am a super being.”)
* What are you capable of? (list the attack capabilities / super powers)

That is, create methods for the above. You could have abstract versions of these that you override in your implementation class.

# Step 2

Now create a BeingGenerator application class to test your architecture. Have it create 2 super beings and a human being. One of the super beings should transform to or from the human being. It should look something like the following, though it may differ in some ways depending on your approach to step 1, so it is indicative only:

**public** **class** BeingGenerator {

**public** **static** **void** main(String[] args) {

List<BaseAttack> attacks = **new** ArrayList<BaseAttack>();

attacks.add(**new** Punch(3));

attacks.add(**new** EyeLaser(1));

attacks.add(**new** FreezeRay(1));

Being megaMutant = **new** SuperBeing("Mega Mutant", **new** CoreAttributes(20, 15, 12, 24), attacks);

attacks = **new** ArrayList<BaseAttack>();

attacks.add(**new** Kick(1));

attacks.add(**new** Punch(1));

Being bruceBanner = **new** HumanBeing("Bruce Banner", **new** CoreAttributes(5, 3, 4, 7), attacks);

attacks = **new** ArrayList<BaseAttack>();

attacks.add(**new** Kick(4));

attacks.add(**new** Punch(5));

Being hulk = **new** SuperBeing("Hulk", **new** CoreAttributes(20, 15, 12, 24), attacks, bruceBanner);

hulk.whoAreYou();

hulk.whatAreYou();

hulk.whatAreYouCapableOf();

megaMutant.whoAreYou();

megaMutant.whatAreYou();

megaMutant.whatAreYouCapableOf();

bruceBanner.whoAreYou();

bruceBanner.whatAreYou();

bruceBanner.whatAreYouCapableOf();

}

}

Your output might look something like:

I am Hulk

I am a super being!

I have the following attacks:

- Kick [x4]

- Punch [x5]

I am Mega Mutant

I am a super being!

I have the following attacks:

- Punch [x3]

- Eye Laser [x1]

- Freeze Ray [x1]

I am Bruce Banner

I am a human being!

I have the following attacks:

- Kick [x1]

- Punch [x1]

# Step 3

Create a new application class (or replace BeingGenerator) that generates random beings (names, types, attributes, etc. aren’t that important) and stores them in files in a shared folder. Limit yourself to 10 for the moment.

I recommend making your classes Serializable and you could use ObjectOutputStream to create files, such as being-0001.ser, being-0002.ser, etc. The main thing is unique filenames.

See <https://www.tutorialspoint.com/java/java_serialization.htm> for an example.

You should probably use something like Thread.sleep() to slow things down a bit so that the beings and their files are created periodically rather than in a single glut.

# Note on future lab documents (related to assignment 1)

The next lab document will ask you to modify the application class from step 3 to make use of a factory class. This approach will mean you won’t have to directly create SuperBeings or HumanBeings with the *new* keyword.

You should also implement other design patterns where appropriate, such as singleton.

You will also create an observable class to monitor the folder for new beings. Another observer class will be notified when new beings arrive and will use threading to initiate battles between 2 beings – obviously in a battle royale, there can be multiple simultaneous fights to the death. You can come up with whatever algorithm you like to simulate a fight between 2 beings (using their attacks). You should probably include some random element that gives a human an occasional chance of beating a super being.

Depending on how things go, you might send the survivor of each fight back into the arena (the shared folder), to be added back to the list of beings to be paired up, and on it goes till only 1 remains. We may not get that far, though.

# **Note on Plagiarism**

**I need you to consider this very carefully and seriously. I have given you enough flexibility in what you code that plagiarising the code of colleagues is almost certain to be caught and immediately result in a zero grade for all concerned – both givers and receivers. I am very good at spotting code that has been copied and then modified to give it the appearance of originality – I have been setting these types of assignment and correcting them for years, so I know all the tricks. Don’t take the chance!**